SOFTWARE MAINTENANCE: RESEARCH AND PRACTICE, VOL. 9, 177–200 (1997)

Interview

Interview with David E. Peercy

DAVID E. PEERCY[1]* AND NED CHAPIN[2]

[1]*Sandia National Laboratories, Box 5800, Albuquerque NM 87185-0638, U.S.A.*
[2]*InfoSci Inc., Box 7117, Menlo Park, CA 94026-7117, U.S.A.*

SUMMARY

David E. Peercy summarizes and explains his more than 20 years of risk assessment work on software maintenance, including an analysis of more than 80 major systems. He places software maintenance in a larger context of software support and integrated logistic support, and describes the process of risk assessment. He contrasts his view of risk with common governmental, management and auditing views, and with the positions of others in the field. Drawing upon his findings and experience, he closes with four major insights about software maintenance, emphasizing the importance for an organization of having supportable software.

J. Softw. Maint., **9**, 177–200 (1997)

No. of Figures: 1. No. of Tables: 2. No. of References: 31.

KEY WORDS: risk assessment; risk management; software support; software logistics; software supportability; software maintenance

1. BIOGRAPHY

David E. Peercy is a Senior Quality Engineer at Sandia National Laboratories in New Mexico leading software evaluations and analyses for major software systems. Since joining Sandia in 1990, he has developed processes for the qualification and acceptance of software products to satisfy the Department of Energy (DOE) Software Quality Assurance Inspection Procedure (QAIP). Also, he has participated in four major Sandia programs: the Software Management Program (SMP) Quality Improvement Team (QIT), the SMP Software Process Assessments Program, the Software Quality Assurance Subcommittee, and the Software Reliability Working Group. In addition, Dave has served as Sandia's technical lead on Cooperative Research and Development Agreements with SEMATECH on the Software Process Improvement project, and with General Motors AC Rochester where he initiated a Software Process Improvement effort. Dave is currently the chair of the Society of Automotive Engineers G-11 Reliability, Maintainability,

* Correspondence to: David E. Peercy, Sandia National Laboratories, Box 5800, MS-0638, Albuquerque, NM 87185-0638, U.S.A. E-mail: depeerc@sandia.gov

CCC 1040–550X/97/030177–24$17.50

Received 8 September 1996
Revised 20 February 1997

Supportability, and Logistics Software Committee. Dave recently became certified by the American Society for Quality Control as a Software Quality Engineer.

From 1977 to 1990, Dave served as technical lead for the development of a practical Risk Assessment Methodology for Software Supportability (RAMSS) for the U. S. Air Force Operational Test and Evaluation Center (AFOTEC). During this period, Dave was also technical lead for a Computer Security Enhancement Review project, a Secure File Transfer Program, a security analysis project for a Strategic Defense Initiative program, a Command Experimental Vehicle (EV) program, an evaluation of an Automated Message Processing Exchange product, a concept definition for a life cycle computer system security test and evaluation project, and several Pascal and Ada projects. Dave was a member of the ANSI X3J9 Pascal standardization committee. He has conducted courses on Ada, Pascal, software logistics (sponsored by the Society of Logistics Engineers (SOLE)), software inspections, software measurement and software process improvement.

Dave has seven publicly available reports published, and 13 papers in serials and conference proceedings. Of these 20 items, 12 (60%) are on aspects of software maintenance. In addition, Dave has made 19 presentations at conferences and symposia, of which 14 (74%) were on aspects of software maintenance. Of Dave's more than 22 major project reports, about a third include some specific coverage of software maintenance topics.

Dave earned his B.S. degree in Applied Mathematics at the University of Colorado in 1966, and his M.S. degree in Mathematics at New Mexico State University in 1967. In 1971, he completed his Ph.D. in Mathematics at New Mexico State University. This interview was conducted by an exchange of e-mail and letters in late 1996 and early 1997. Dr. Peercy's e-mail address is: depeerc@sandia.gov

2. INTERVIEW

Ned: Dave, how did you get involved in software maintenance?

Dave: In 1977 I moved to a position with BDM Corporation in Albuquerque, NM. My first project was a task for the U. S. Air Force Test and Evaluation Center (now called the Air Force Operational Test and Evaluation Center—AFOTEC). The objective was to rework and enhance a preliminary set of factors developed by AFOTEC to measure the maintainability of software systems being developed for the Air Force. In 1977, AFOTEC had the responsibility of assessing complete systems during a project phase that had come to be known as 'Operational Test and Evaluation'. Realize that in 1977, most systems being developed or supported (maintained) did not have a large functional dependence on software. The Air Force was perceptive enough to know that:

(1) software use in systems was increasing;
(2) software was beginning to be a costly component;
(3) much of the software cost seemed to be in the maintenance activity; and
(4) the Air Force had no systematic method to determine whether software in its acquired systems could be maintained.

In 1977, AFOTEC system evaluations were in terms of effectiveness (operational function of the system) and suitability (capability of the system to be supported

and available when needed for wartime function). The software evaluations had been pretty much *ad hoc* without a real process in place or stable criteria against which a repeatable/reliable assessment could be conducted. I was assigned to lead the task focused on software suitability where AFOTEC's primary concern was software maintainability.

Ned: What was the basis for this work on 'suitability'?

Dave: The concept was to develop a method to assess the software's maintainability by reviewing documentation and source code characteristics during Operational Test and Evaluation. The objectives were to identify which characteristics were important for maintainability and develop an efficient and repeatable process for measuring these characteristics. This task was the beginning of a series of many tasks over approximately 12 years that eventually led to a comprehensive methodology (process, criteria, tools) that enabled AFOTEC to assess the risk to the Air Force of supporting a software system. This comprehensive methodology is called the Risk Assessment Methodology for Software Supportability (RAMSS).

Ned: 12 years should have allowed you to get into things in depth. How did you start?

Dave: The 12 years can be partitioned into four major research efforts that were essentially sequential:

(1) development of a framework for product (documentation and source code) maintainability evaluation;
(2) development of a framework for support environment evaluation;
(3) development of a framework for support process evaluation; and
(4) development of a risk assessment method based on the model frameworks, and software maintenance and evaluation data from major systems.

The field work necessary to gather evaluation and maintenance data included visits to all the various Air Force logistic support centres across the United States, the North American Aerospace Defense Command centre in Colorado Springs, MITRE Corporation, National Security Agency, and many more that I cannot possibly remember. We focused on more than 80 systems including nearly every Air Force military-related system active at the time, such as aircraft operating flight programs, weapons, radar systems, test equipment and general support for Air Force systems. We concentrated on process, product, and environment factors and characteristics.

Some of the work used a research approach, where information was gathered through surveys, expert opinion, literature search, and so forth. This work resulted in the establishment of the basic framework of major factors and characteristics for RAMSS. Some of the early work in 1977 was based on the work of Thayer (1978), who worked at AFOTEC, and who was the initial investigator for the Air Force in this effort. Other literature that played a major part in the early work included a treatise on software quality characteristics by Boehm *et al.* (1978) and the early work on factors in software quality by McCall *et al.* (1977) for the Rome Air Development Center (RADC, now called the Rome Air Center). We did a lot of statistical analysis of the maintenance data collected during our

visits to the various software support centres. And, of course, there were many Air Force personnel and BDM staff members who contributed to this development effort in addition to myself.

Ned: Your discussion appears to make software 'supportability' the same as, or perhaps another name for software 'maintenance'. Could you please clarify some of your terms for me?

Dave: I have used the terms software logistics, software maintenance, software maintainability and software supportability. Software logistics is the selective application of the integrated logistic support process to the software components of a system. Since the integrated logistic support process is concerned with system support throughout the entire system life cycle, there is a connection to software support and software maintenance.

Briefly, *software maintenance* is the set of activities conducted to incorporate change requests into the software product after the software product is delivered to the customer. The change requests are typically to correct defects in the software, add new enhancements through functionality or performance, substitute new functionality or performance for old, or adapt the software to changes in the operational environment equipment or data interfaces. In reality, these activities begin to occur prior to official delivery to the customer.

Software support includes software maintenance activities as well as many other activities that occur during the support phase, such as handling customer service requests, packaging and delivering new releases, installing the software, conducting beta tests with customers, and so forth.

Software logistics is still more inclusive. It is a comprehensive approach to managing software support through the complete life cycle of the software and its associated application systems. In full form, it is called integrated logistic support.

Ned: Could you diagram those relationships?

Dave: Sure can (see Figure 1 and Peercy (1994) for more information).

Software maintenance takes as little as 30% and as much as 95% of the software support time and budget. Support for the customer, training, help desk servicing, delivery, installation and configuration management of the software product at customer sites are all examples of tasks that are software support but generally should not be considered to be strictly software maintenance. All the factors and characteristics identified in the RAMSS development apply to software maintenance. The collected software maintenance data on over 80 systems specifically identify types of systems, number of changes, complexity of changes, priority of changes, maintenance effort and schedule information for each software maintenance block release.

I would like to come back to these terms later. I have evolved (and continue to evolve) my understanding of these terms over many years (Peercy, 1996; AIR5121, 1997). Perhaps sometime during our discussions I can clarify further the relationships of logistic support, software support and software maintenance.

Ned: In addition to those terms, you used the term 'risk'. Does that mean you were concerned with security matters such as guards, passwords, dual disks, mantraps, firewalls, etc.? Could you give me some background on what you mean by 'risk'?

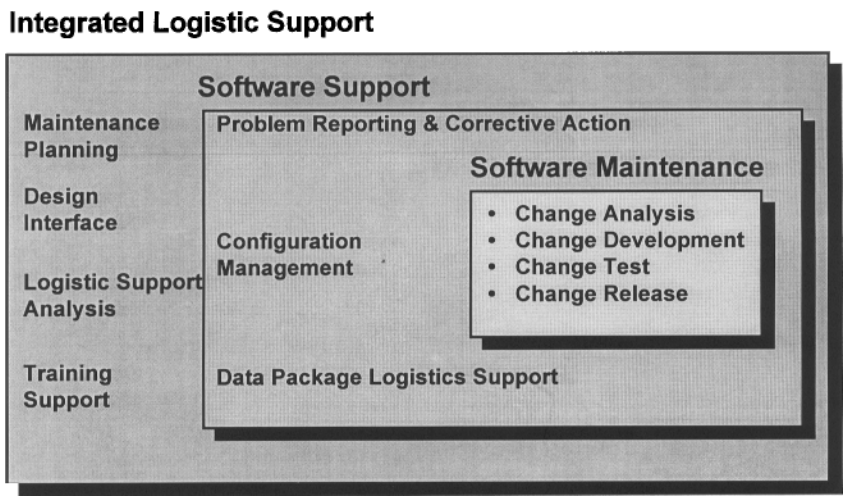

Figure 1. Software support encompasses software maintenance, and integrated logistic support encompasses software support

Dave: Well, since I brought it up, I guess it is only fair that I should clarify what I mean by 'risk'—in particular, as it relates to software supportability. Risk is the potential for the realization of unwanted, negative consequences of an event (Rowe, 1988). There are many questions that arise from this definition as we attempt to apply it to a particular application. For example, what do we mean by the words 'potential', 'realization', 'unwanted', 'negative', 'consequences' and 'event'? Nearly every word of the definition is an avenue to explore. Of course, we don't just want to know that there are or may be negative consequences—we also want to know how to prevent or reduce them. This means that we will not be satisfied with just a final risk number. We must have details as to what characteristics might best be improved to get the greatest risk reduction.

Processes related to risk that provide a structure for me include risk determination, risk evaluation, risk assessment, and risk management. I tend to use the term risk determination to mean the process of identifying and estimating the magnitude of risk. Risk evaluation is the process of developing acceptable levels of risk to the risk agents—individuals and societal groups to whom the consequences go. Risk assessment is the combined process of risk determination and risk evaluation over time, and may include some iterations. The total process of identifying, controlling and minimizing uncertain events and their unwanted consequences is risk management.

Software supportability (as used in the RAMSS) is primarily concerned with risk assessment. Software supportability is a measure of the adequacy of products, resources and procedures to facilitate the support activities of modifying and installing software, establishing an operational software baseline, and meeting user requirements for system availability (Peercy, Tomlin and Horlbeck, 1987). Risk determination is concerned with the negative outcomes that are the result of

inadequacy in products, resources, or procedures to accomplish the support activities in an acceptable manner. Risk evaluation is concerned with defining what 'acceptable' means relative to the responsible person or group and the baseline probability density function of expected maintenance activity. A required maintenance action is not necessarily negative. Too many required maintenance actions, or the inability to complete a required maintenance action within a specified period of time, may be negative depending upon what is termed acceptable. Note that there may be more than one interpretation of what is 'acceptable' depending upon the responsible group. For example, the user and supporter may differ in the interpretation of the consequence of missing a particular release deadline.

You also asked whether I was concerned with security-related prevention mechanisms. The answer is yes, if the security-related mechanisms affect the supportability of the software. It is more difficult to support software that is classified. It is more difficult to support software that has been developed in accordance with security requirements, such as formal specifications, trusted computing base, vulnerability analyses, specialized security testing, and so forth. Although such rigour may result in a trusted software product, the product must remain trusted after changes are made. My experience is that very little planning is done to ensure the processes and environment are in place to adequately support trusted software.

Perhaps the key to understanding risk terminology is to realize that risk is relative—relative to whoever or whatever is the recipient (the risk agent) of the consequences. What may be extremely risky for you may be of little risk to me. Risk is also dependent upon 'time'—that is, a risk agent bears or assumes a risk for some particular calendar time.

Ned: So is 'risk' relative, and context and participant dependent?

Dave: Yes. Risk is most often related to one of project management's big three: schedule, cost and performance. For software supportability, the next scheduled release might be an event of interest. Possible attributes of that event might be the release date, cost and quality (e.g., lack of software defects). Unwanted, negative consequences might range from customer irritation at missing the release date by a day, to a patient death due to failures in medical equipment from defects in the released software product. The demise of a company may be attributed to a late software release that causes the company to miss a critical market window. The crash of a major airliner may be attributed to a software defect that causes the aircraft to automatically descend 2000 feet (watch out for that mountain!) when the wings begin to ice up. In reality, many unwanted consequences cannot be traced to a single event. A very difficult task in risk determination is quantifying the relationship of the unwanted consequences of an event with multiple points of failure. Since risk is a quantification of the relationship of unwanted consequences to the potential for occurrences of some causal or correlated events, you can see how difficult it might be to quantify software supportability in terms of risk.

Risk is a derived measure, dependent on inherent characteristics/measures of event attributes. For software supportability, the inherent characteristics and measures are derived from the processes used to develop and support the software and

its encompassing system, the products (e.g., documentation and source code) produced by those processes, and the environment (e.g., personnel, tools and systems in which the software is supported). Through my own experience, I have come to understand the importance of software support within the system context and the relationship of software maintenance to the more comprehensive characteristics of software supportability, and the view of software supportability within the complete integrated logistic system support concept or software logistics. The RAMSS focus is software supportability, which includes characteristics of software maintenance but not necessarily all the characteristics inherent with the integration of software support within a system logistics framework. The RAMSS focuses on the risk that existing or planned facilities and resources will not be able to accomplish satisfactorily the software changes required during the operation and support phase of a system.

Ned: How about some examples of 'risk' going from simple to realistically complex?

Dave: That's a great idea! I'll try to capture the ideas of risk through some examples—all related to software supportability and maintenance. I'll first describe some simple software examples where software support/maintenance has 'risk'. Then, I'd like to give an example of software support and risk using the risk processes I mentioned earlier. Finally, I'd like to provide an example of a complex software system and some of the major aspects of software supportability risk, primarily in terms of the RAMSS approach developed for AFOTEC.

So, to my first 'example'. Many application areas have significant software support risk. These applications include medical treatment, banking information, network communication and military mission critical. In most systems the software support risk derives from the criticality of the software. Software must operate dependably and software updates (modifications) must be defect free and delivered in a timely manner so the operational mission of the system can be accomplished.

Ned: That sounds like a lot of systems. Can you be more specific?

Dave: Consider a medical treatment system that via software controls administers radiation treatments to cancer patients. Malfunction of this software could result in a radiation overdose and cause serious injury or death to the patient. We typically focus on the operational correctness of the software, but rarely seem to realize that a primary problem with the correctness is going to occur after changes are introduced to the software during its support. We must keep the software operating correctly when functional and performance enhancement changes are made and adaptations to new and improved equipment are implemented. These changes are sources of software support risk. If the software is not designed for such changes, or the support process does not provide an adequate implementation, the patient may be at risk.

Consider the software support risk in a banking information system that performs electronic funds transfer. If changes to the software are not correctly made, billions of dollars may be incorrectly transferred to nearly anywhere. Interest lost from the incorrect transfer, even if corrected in a timely manner, can amount to hundreds of thousands of dollars. Banks and receiving corporations can lose business or even go out of business. Bank transaction systems are on-line most of the time. This presents little opportunity to test adequately the software changes

in an operationally similar environment. In some instances, test files have been left in the transaction queue and inadvertently processed as valid funds transfers.

Ned: How about some additional situations?

Dave: The Internet is a communications nightmare for software support risk. Software drives the network flexibly, but the network is rapidly evolving and the logged-on users continue to increase in number and diversity. Just imagine the risk of having an unwanted negative consequence propagated throughout millions of network nodes and attached computer systems! Network software that has been changed is a prime source for such propagation risk. Blackouts of communication networks due to incorrect software changes have been documented. Software that has not been designed for security protection, or whose designed security protection has not been correctly reverified when the software is changed, presents great opportunity for virus propagation.

A military system such as the U. S. National Missile Defense system relies on 24-hour-a-day and 7-days-a-week availability and presents many software supportability risks ('challenges'). Emergency responses when the system goes down require immediate maintenance, test and installation servicing—in zero real time, at least it probably seems like that. Personnel work around the clock and become tired and prone to mistakes. As in the banking example, testing on the actual operational system is limited. The software support environment requires construction of a costly emulation capability that in itself provides some risk that the emulation may not match the operational system. Results of software support inadequacy can and have resulted in some rather tense (and verbally terse) national situations. Catastrophic events, such as invalid missile launches, or not launching missiles when necessary, are among the risks that might occur from inadequate software support/maintenance.

Ned: That example covered a lot of ground for a 'first example'. What is your second example?

Dave: For a second example I'll walk through the framework of risk processes I introduced earlier. In this example, vendors provide software support servicing, and my company's internal computer support organization accomplishes replacement of commercial software on the networked PC workstations. In this example it is reasonably easy to understand some aspects of the risk, how the risk might be computed and possible actions to take based on the resulting risk evaluation.

(a) *Risk agent:* I am a user of the system, and as such will assume the role of risk agent in this example—how am I affected by the support?

(b) *Risk determination:* this is a measure of the inconvenience to me while my computer service group replaces software. This measure will be determined by how much time it takes to replace a software package, how often a software package is replaced, and whether each replaced software package has performance equal to or better than the replaced package—including access to previous work products. Assume the information below (don't put too much significance on the numbers) as it relates to the current commercial packages on my computer system:

- number of packages = 20,
- average number of replacements per year = 2,
- average time to perform replacement = 2 hours,
- index of replacement effectiveness = 0·9; (means for every nine hours of lost time due to a replacement, there will be one more lost hour because the replacement hampers my work).

Risk measure = total lost time due to replacements per month/total available work time per month

Ned: I note that time per month cancels out since it is in both the numerator and denominator of the ratio. So then, does the risk measure come out as a dimensionless number, in the same way that the cyclomatic number (McCabe, 1976) is a dimensionless complexity measure?

Dave: Yes, risk is dimensionless as the potential for the unwanted consequences of an event. However, those measures from which risk is derived may not be dimensionless. In the example as you noted, the risk measure is derived from two measures that have time as a dimension. May I talk about risk units a little more later on? To illustrate the risk determination example we are discussing, suppose the following data have been collected over several months:

- total lost time due to replacements (per month) = $(20 \times 2 \times 2/12) \times (1/0{\cdot}9) \sim 7{\cdot}4$ hours,
- total available time for work (per month) = 152 hours,
- risk measure = $7{\cdot}4/152 = 0{\cdot}049$; (loosely, this risk measure is the potential for losing time at any instance during the normal work hours).

(c) *Risk evaluation:* this is the process that derives the levels of risk and the actions to take based on those levels. Assume the evaluation process has established the risk levels and subsequent actions (see Table 1).
Note that a running three-month average of the monthly risk measure could be plotted for each month and then used as the measure for decision making.

(d) *Risk assessment:* this is the combination of risk determination and risk evaluation, and provides concurrent co-ordination of the tasks. The assessment

Table 1. Risk levels and action alternative

Risk measure	Risk level	Risk action
0·00–0·04	Low	None
0·04–0·08	Medium	Ask computer support to perform some updates during non-working hours
>0·08	High	Submit complaints to management software vendors whose packages are being updated too frequently or have too many failures

process may iterate over time to evolve the risk determination and risk evaluation processes in support of risk management. In other words, I may adjust the risk levels and my specific actions over time so as to improve my productivity.

(e) *Risk management:* these are my own actions as I try to minimize the risk measure. Table 1 summarizes a few of the actions I might take for different levels of risk. Since risk determination in this example gives a risk measure of 0·049, my risk is at the Medium level. Table 1 dictates that I contact the computer support personnel and work out a way that they could perform some of the software updates during hours that I am not working. The risk evaluation process should already have identified which specific support actions are possible and what is to be done at each risk level. This is not the same as procrastinating until, with the software in use, the risk becomes 'too high'!

Ned: That second example made the concepts hang together better for me. What does your third example offer?

Dave: Are you ready for a third example? OK, let me take one involving software support for a new military missile system. These days, most large new complex hardware products, such as commercial aircraft, subway trains and military missile systems incorporate significant software components. In this particular case, let us assume the military desires to adopt a reasonable strategy to reduce the risk of inadequate software support.

There must be some agreement as to what levels of risk require specific responses from the risk agents, so that I could prepare situation–action pairs, like Table 1 or the equivalent. There are essentially three risk agents (acquisition, user, support) that must balance their own perspectives to achieve agreement as to what is adequate and what is not adequate software supportability. The acquisition agent is concerned about getting the software developed on time within budget with the requisite quality. Software supportability is part of the requisite quality. However, if it takes time and money to build in software supportability characteristics this activity is frequently viewed as outside the scope of acquisition funding. The user agent is concerned with availability of an operational system for all possible operational scenarios, including changes to the software that may be necessary to support the required system's effectiveness.

Software support must ensure that an operational software version is installed on the missiles and satisfies the required mission, wartime or peacetime. Timeliness of response to software problems and ensuring software updates accomplish the intended purpose (correct a defect, provide an enhancement or adapt to environmental change) are of concern to the user agent. The support agent is concerned about having adequate processes, environment resources (personnel and systems) and software product characteristics to meet the user agent requirements. The user agent and support agent must agree on what level of support is adequate so that the operational mission(s) of the new missile system can be accomplished.

Sometimes this agreement at the system level is in terms of availability requirements.

Ned: The software used with NASA's space shuttle systems—isn't that an example too?

Dave: Yes, that is a good example where software support risk is an issue, and where more than 'adequate' resources are provided in order to ensure the support can be accomplished without defect. Of course, the shuttle's software support is costly. As you can see by such examples, the concept of 'risk' and its assessment and management can be complicated. There are many risk agents, risk factors and competing priorities. In these examples, some of the software may be embedded within the hardware. So the support needs of software must be integrated with the support needs of the hardware and the associated risk to the operational effectiveness of the system.

The purpose of the RAMSS developed for AFOTEC was to provide a focus on the primary factors and defining characteristics of software processes, products, and development/support environments so that a reasonable software supportability risk assessment approach could be established for software systems, such as in this example. The criteria and measures were defined to assist the software supportability risk assessment process at any point of the life cycle and within the context of a system supportability/Integrated Logistic Support (ILS) approach. The RAMSS approach included early attention to building software to be supportable and developing a software support concept based on agreement between user agents and support agents as to the level of expected support activity. With early software supportability risk assessments it would be possible to assist logistic trade-off studies to cost-effectively reduce software support risk.

Two risk measures are of primary importance: a measure (evaluated risk) that provides insight into whether the software has been or is being built for supportability, and a measure (estimated risk) that provides insight into whether the existing or planned software support resources and environment are adequate to support the expected software change profile during the software's operational use. The RAMSS provides a reasonable way to compute these two measures—but more importantly provides insight into what factors and characteristics of the software go into the computation and are candidates for improvement to reduce software supportability risk.

Ned: I am still confused on the units of measure of risk—is risk always dimensionless? Could you please clarify?

Dave: Risk as a 'potential' can be thought of as a mapping of attributes of the risk-related events to a numeric or symbolic value. Let me express it this way:

$$\text{Risk function} = \{(e,a,c,g,m)\} \tag{1}$$

where: 'e' is an element in the set of events of interest, 'a' is in the set of attributes of the objects to which the events apply, 'c' is in the set of unwanted negative consequences, 'g' is in the set of possible risk agents, and 'm' is the measure or 'risk' for some appropriate measurement scale.

As I confirmed to you earlier, risk is shown by numeric or symbolic values

and has no units. One typical risk measure unit is a probability value between zero and one. For discrete cases, the measure of risk might be the average frequency of an unwanted event divided by all possible events over some time period. For example, when producing wafers in a semiconductor manufacturing facility, all possible events might be computed as the total number of wafers produced over a given time period. The unwanted event of producing a defective wafer can be measured over the same time interval. The risk measure of the unwanted event is the average frequency of defective wafers to total number of wafers produced per unit of time.

Another typical risk measure unit is an ordinal position identified by symbols, such as Low, Medium and High, or such as Condition Green, Condition Yellow, and Condition Red. For example, the system user or customer may decide that the software supportability risk is:

- High—if 50% or more of software modules have cyclomatic complexity numbers greater than 10;
- Medium—if 20% – 50% of software modules have cyclomatic complexity numbers greater than 10; and
- Low—if less than 20% of software modules have cyclomatic complexity numbers greater than 10.

The decision may be to reject software that is High risk, ask for modifications to Medium risk software until the Low risk level is reached, and to accept Low risk software. Note that this is just an *example* of units of risk in terms of the measurement scale—not really a suggestion of what one should do to assess supportability risk.

Probability measures tend to be more precise (at least on the surface) and symbolic enumeration measures tend to be more subjective. The purpose of identifying measures of risk is so that appropriate decisions can be made. It may be that a less rigorous risk measurement unit is adequate (interpret this to mean more cost effective) to support the required decisions.

Ned: Is some paraphrase of Lord Kelvin's famous 1890's statement (Kaplan, 1992, p. 504) the basis for expressing risk in quantitative terms?

Dave: Even earlier is the quote 'What is not measurable make measurable' attributed to Galileo Galilei (1564–1642) and probably is a basis for expressing risk in quantitative terms (Finkelstein, 1982). The version I like best when considering risk says you cannot control what you cannot measure (DeMarco, 1982, p. 3). That argument is fine as far as it goes, but is not complete (Fenton, 1991, p.7). It is not enough to assert that we must measure in order to gain control or manage. Measurement activities must have clear objectives or goals—and as you have correctly surmised, one typical way to relate the measurement and goals is through the measurement of risk.

Ned: Well, that fills me in on some background. Why did 'risk' get to be a significant consideration in your work?

Dave: The culmination of the work with AFOTEC was the development of a Risk

Assessment Methodology for Software Supportability (RAMSS). As the tasks progressed from defining a software (product) maintainability evaluation methodology (Peercy, 1981) to a methodology for evaluating a software support environment (Peercy and Swinson, 1983), it became apparent that there should be some overall framework that covered product, environment AND process. Furthermore, this framework should be validated using 'real data' as a basis. Although it was nice to have numerous measures of the characteristics and factors, it wasn't clear at first to AFOTEC how the evaluation information was going to be used to assist it in meeting its primary objectives. AFOTEC's most important objective was to present a concise summary of supportability recommendations to the command chain that could be defended, as needed, through reference to lower level assessment measures. One recognized way to gain the attention of military commanders is to use the term 'risk', so it was decided to explore the possibility of presenting the support of software as an activity having a measurable risk.

The last phase of the AFOTEC set of tasks was initiated to develop methods for assessing process characteristics (development and support) that affected the supportability of software and to obtain a database of actual maintenance and evaluation data upon which risk measurement could be based. The result was RAMSS (Peercy, Tomlin, and Horlbeck, 1987).

Ned: Let me get some perspective. For that, contrasts sometimes help. One contrast is with the term 'risk' and the use of measures of risk by information systems auditors. Thus, Bill Perry, one of the *Journal's* Advisory Board members, states that risk is the probability of loss (Porter and Perry, 1987, pp. 93–99). Loss occurs when vulnerabilities in systems get converted into the actual incurring of an otherwise avoidable increase in cost or decrease in revenue. An example of a vulnerability in a system is a system's occasional failure to produce correct output. Three examples of resulting cost increases are the loss of raw materials, the diversion of labour to clean up or repair work, and the time needed to rerun to get the expected correct output. In information systems auditing, the usual risk evaluation process involves a quantitative scoring and estimating of the vulnerabilities, the probabilities of losses, and their associated cost increases or revenue decreases for various time intervals. The unit of measure for risk is the local currency, such as dollars, pounds, rupees, etc. How is what you have described in your AFOTEC work different from the information systems auditing approach?

Dave: I've known Bill Perry for some time and respect the many contributions he has made. We participated together in some of the first software maintenance conferences and over the years I've supported his organization's quality conferences. Let me consider your audit example with a parallel RAMSS example so some of the similarities and differences can be explained.

Suppose an audit of a support organization evaluates the potential loss that could result to the organization from vulnerabilities (lack of adequate supportability characteristics) in software product, process and environment. In this case, the potential loss would be due to a software block release event that is not acceptable (is late, has defects, costs more than is budgeted, etc). Three examples of potential losses for the organization are: monetary fine from the customer due to the late

block release; increase in cost due to an increase in personnel to meet the block release schedule; and, cost due to rework required to remove an unexpectedly high level of defects found during the block release. RAMSS addresses the *estimated risk*—the potential for missing a block release schedule in the presence of some vulnerabilities, such as personnel level and maintenance workload level. In addition, RAMSS addresses a related 'quality-like' *evaluated risk*—potential for impacting the block release schedule and/or defects in the presence of vulnerabilities in the form of inadequate supportability characteristics that might either contribute to or prevent losses from a block release. RAMSS addresses the potential loss due to increases in personnel and rework indirectly through risk evaluation trade-off studies.

This example illustrates some of the similarities of RAMSS to the approach used by information systems auditors. However, there are some major differences. The RAMSS algorithms for computing estimated risk and evaluated risk are determined relative to the historical evaluation data and maintenance data from a particular set of 80 systems. As far as I can tell, the audit algorithm for computing potential for loss is not based on an historical database and is not a correlation equation based on statistically significant vulnerability factors. The audit algorithm is a simple summation (over all known vulnerabilities) of each vulnerability probability times the potential loss due to the vulnerability. This summation results in a dollar value that represents the loss part of the risk. The RAMSS estimated risk and evaluated risk are dimensionless potentials for loss, not the loss value itself. The loss information and other cost indicators are embedded within the parameters of the risk equations. An acceptable level of risk can only be defined by the two primary risk agents: user and supporter. Loss in some currency unit would be relative to an estimated or evaluated risk level that is higher than the specified acceptable risk levels.

In information systems auditing, saying that the unit of measure for risk is the local currency might be a slightly inaccurate variation on the terminology, but is not a particularly significant difference. We frequently describe risk in terms of the magnitude of the loss, but the units of risk are still the dimensionless 'potential' and not the units of the potential loss value itself. We might say the risk is 'high' of losing $25 000 or the risk is 'low' of losing $25 000. Or, we might say the potential loss due to the specified vulnerabilities, probability of occurrence and associated dollar loss with each occurrence is $25 000. In none of these situations are dollars the unit of risk, but risk is explained in terms of the potential loss, which *is* in units of dollars. Remember that risk is the *potential* (dimensionless) for unwanted negative consequences (e.g., loss in dollars) of an event (vulnerability incident with estimated probability of occurrence). To be entirely accurate we should carefully delineate between the risk measure and the potential loss itself.

Ned: That was helpful. Let's try for another contrast, this time with the 1980's 'quantitative risk assessment (QRA)' and the more recent 'partitioned multiobjective risk method (PMRM)' approaches, used for example by the U.S. Federal Government Environmental Protection Agency (EPA) (Haimes *et al.*, 1990). The

EPA expresses risk as the expected loss from the relevant hazards. Basically, this is, for some time period, the product of the probability of the event times the net incremental cost arising from the hazards. In QRA, including categories of situation variations leads to different risk amounts that are summed for the overall risk. For PMRM, the probabilities and costs are averaged. For both QRA and PMRM, the unit of measure for risk is the local currency. How is what you developed in your AFOTEC work different from the QRA and PMRM approaches?

Dave: The risk methods you have described for this contrast are similar to the auditing I just covered. Basically the risk measure is computed as a potential for loss due to multiple vulnerabilities/hazards, each of which has an associated probability of occurrence and loss value. The algorithm for computing the potential loss is similar to the information system audit approach. The algorithm differs from the RAMSS computation as I mentioned previously. The specific underlying attributes that are measured, the methods and techniques used to conduct the evaluation and compute the expected loss measures, and the meaning of the risk for RAMSS applications is different from the QRA and PMRM approaches. I've tried to envision how the QRA and PMRM approaches might be applied to determine software support risk (potential for loss due to an unacceptable block release) as determined by RAMSS, and it is not clear that these methods are directly applicable.

Ned: So you are not only pointing out differences, but also questioning the applicability. Does that mean that what you did for AFOTEC is only applicable to military systems?

Dave: Well, the applicability of the other risk methods to determining software support risk would be difficult, so I am questioning their applicability. The cross dependencies of the many software supportability characteristics are very difficult to understand. Even with all the work and effort put into RAMSS, there are definite limitations in its applicability. The limitations are not necessarily due to the methods, principles or general framework derived or the application domains (e.g., military, business systems). Most of the limitations are due to the accuracy and applicability of the historical data upon which the risk measure mappings were derived.

RAMSS is definitely applicable to military systems and related systems that could be similarly categorized. RAMSS should not be directly applied to business information systems without having developed a database of actual maintenance data and evaluations from similar systems (or at least a database of archetypical data for temporary use) upon which to derive the risk measure mappings. Such a database could be derived for specific companies and application-specific areas. The data would need to be analysed in a manner similar to that which was done for AFOTEC and an appropriate factor analysis conducted. I do believe that the top level framework for product, process and environment would still be validated by the factor analysis. The lower level supportability characteristics and correlation equations for the risk measure mappings would probably be somewhat different.

Of course, RAMSS is now getting to be pretty old, even for use on military systems. I still use parts of the methodology for internal evaluations and for

coercing anyone who will listen (and many who won't listen) into building software to be supportable. The factors and characteristics that were identified and even envisioned back in the late 1970s and through the 1980s are still significant, but there are many other characteristics and new thoughts on software engineering that are also important. Software engineering paradigms, such as information hiding were built into the product maintainability characteristics. Most of the Ada language design rationale and environment characteristics were built into the product maintainability characteristics and support environment characteristics before Ada was even named. Some of the process-orientated issues that are now part of models such as the Software Engineering Institute's Capability Maturity Model key process areas (Humphrey, 1990) are built into the process characteristics—with a supportability viewpoint. Still, there is so much more to software supportability and the broader area of software logistics that could be and should be covered in order to get a good perspective on software supportability risk.

Ned: May I ask you to draw upon your diverse experience in software maintenance work for some views on some related work, please? Do you know the book by Marian Myerson entitled *Risk Management Processes for Software Engineering Models*, published in 1996 by Artech House, Inc.? That book seems to cover quite a bit on topics like industrial espionage, process maturity models, security threats and security generally. What, if anything, is there in this book on risk management relevant to software maintenance, in terms of risk as you have talked about it?

Dave: Myerson's book does cover quite a range of topics with a focus on the security risks and process risks as they relate to software engineering models. The book covers almost nothing about risk as it relates to software maintenance or support characteristics, although the 'operation and maintenance phase' is mentioned with some of the software engineering models. The book seems to be focused mostly on development with perhaps the implication that everything applies to maintenance as an extension of development.

One of the most difficult aspects of software support, especially when the support activity is transitioned to a supporter that is different from the developer, is to integrate critical speciality processes (for example, security and safety) with the more usual management and engineering maintenance processes. Since the book does have a focus on security risks, it would have been useful to provide some insight into the additional issues that must be addressed as the software is changed during support.

Ned: What about the earlier collection of papers edited by Barry Boehm and published by the IEEE in 1989 entitled *Software Risk Management* (Boehm, 1989)?

Dave: Boehm's book is a compendium of papers covering areas in Boehm's software risk management hierarchy: risk management is composed of risk assessment and risk control; risk assessment is composed of . . . and so forth. This hierarchy has a little different terminology but is similar to the simple risk processes I introduced earlier. There are at least two papers from the book that have direct application to the RAMSS work in which I was involved.

The first paper of interest is just a reference in the bibliography to a foundation paper by Parnas (1979) on 'Designing software for ease of extension and contraction'. This paper established the important connection between information hiding and ease of software maintenance. Unfortunately, there is no discussion in the body of the book through other papers or writings that links this to software maintenance risk. The paper by Parnas was a significant reference for the software maintenance characteristics and rationale applied to the early RAMSS work in deriving a software maintainability product framework.

The second paper is the Air Force Systems Command/Air Force Logistics Command (AFSC/AFLC) Pamphlet 800–45 published September 30 of 1988 (AFSC, 1988), and contained in pages 148 to 171 in Boehm's book. This pamphlet describes software risk abatement processes and specifically considers and discusses software support using terminology very closely related to RAMSS. In fact, some of the information in the pamphlet has been derived from the research work done for AFOTEC. Chapter 4 of this pamphlet is dedicated to software support risk. This chapter contains the statement:

> 'A key element of support risk is recognition that supportability is important to both the user and supporter of software. Any risk assessment methodology that ignores the interests of one of these parties may escalate a risk that is unacceptable to the other. One way to avoid this is through use of a user/supporter software support baseline that, according to AFR 800–14, is documented in a computer resources life cycle management plan. . .'

This statement and much of the other information (such as AFR 800–14) seems to be directly from the AFOTEC work. Figure 7–3, 'Team Functions', on page 169 describes the risk assessment steps and associated team functions for a sample use of risk abatement. I believe the steps and functions are derived directly from the work for AFOTEC. Paragraph 7–10 on page 171 mentions the AFOTEC work and provides a contact (at that time) for further information:

> 'Software support risk models are in their infancy. The Air Force Operational Test and Evaluation Center (AFOTEC) is doing some pioneering work in this area. . . .'

Ned: I recall seeing something by Sisti and Sujoe (1994) from Carnegie Mellon's Software Engineering Institute, called SRE for 'software risk evaluation'. How does that fit with your work, and what, if any, is its relevance for software maintenance?

Dave: Interesting that you bring up the SRE. I was a member of an SEI–lead team in April of 1994 that used this software risk evaluation method to evaluate a major internal Sandia project on which I was the software quality representative. The SRE is based on a risk taxonomy structure covering Product Engineering, Development Environment, and Program Constraints and uses an evaluation method similar to the SEI software process assessment that is based on the SEI CMM. At the time, the SRE risk taxonomy was not directly linked to the CMM. The SRE risk taxonomy has a couple of elements that are related to software maintenance. Under Product Engineering, Engineering Specialities is the term Maintainability. Under Development Environment, Development System are the terms System

Support and Deliverability. The primary focus of SRE is on establishing a framework within which projects can address software-related risk, and strengthening the ability of organizations to evaluate and manage software-related risk. Software maintenance and support could have some visibility if the perceived risks to the project were identified. The SRE does not specifically target this area and generally is not closely related to the RAMSS effort.

Ned: What about the recent paper by Hareton Leung in the *Journal of Software Maintenance: Research and Practice* under the title 'A risk index for software producers' (Leung, 1996)?

Dave: The paper by Leung could have some interesting relationships to the RAMSS work. This paper presents a good concept of the customer and the related services and support activities that are a part of processing problem reports. The paper focuses on a 'producer risk index' and a 'user risk index' that clearly acknowledges that software risk is dependent upon the risk agent (producer or user). The two risk indexes are computed in the standard manner as a product of an unwanted event probability times impact. In this case the event is a software failure and the impact is dependent on the risk agent's expected loss given a software failure occurs. The probability of failure is estimated from a failure prediction model based on a number of source code complexity metrics. The impact is computed from a set of cost parameters based on the activities related to processing problem reports and handles complex, module-interdependent problems.

The RAMSS depends on estimating a user/supporter agreement for the expected changes that will be processed over a profile of block releases. The changes are characterized in terms of type (correction, enhancement, adaptation), complexity (high, medium, low) and priority (emergency, normal). The Leung paper provides one method of estimating the corrective maintenance actions and complexity that might be expected over the period of time of the block releases. This would help establish a part of the user/supporter agreement. The recognition of the user and supporter interaction by Leung is very significant and clearly reflects concerns similar to RAMSS.

Ned: What about the book by Susan Sherer entitled *Software Failure Risk: Measurement and Management* (Sherer, 1992)?

Dave: I only have limited information on Sherer's book, primarily from a recent conversation I had with Susan about her work and, in particular, the contents of the book. The book provides a detailed explanation of a methodology for assessing where risk exists in a software system due to two aspects: potential for failure in various components of the system, and exposure to the organization in terms of cost if a failure occurs. Susan's dissertation work in 1988 was a complete exposé of this method. The failure risk determination is computed from reliability models such as John Musa's well-known reliability growth model (Musa, *et al.*, 1987). The exposure (loss) is determined from a system/software hazard and fault tree analysis based on system operational profiles specific to the organization, and the effect on the organization if the hazard/fault were to occur. The method is used to direct the organization to make more informed cost/benefit decisions in order to reduce risk. Susan says that the method can also be applied during the

software support phase to direct which software maintenance actions should be conducted. It is a cost/benefit preventive maintenance approach to identifying which corrective, perfective and adaptive changes should be incorporated.

This concept of providing measurable criteria upon which to base software support decisions to improve the software support characteristics is similar to what I attempt to do with the computation of the RAMSS *evaluated risk*. I was intrigued with the use of fault tree analysis and failure modes, effects and criticality analysis methods, and hazards analysis of the software component of a system. I have long advocated that this type of analysis can be a very important design/redesign influence as a part of software logistics analyses conducted throughout the software's life cycle. The use of the reliability modelling to determine the probability of failures along with the identification of potential failures might have some applicability to estimating corrective, enhancive and adaptive maintenance changes that define the user/supporter block release profile needed to compute the RAMSS *estimated risk*. I've asked Susan to bring me up to speed on her work. Thanks for bringing her work to my attention.

Ned: No problem. May I now ask you to be reflective? From all the software maintenance work you have done and seen, you have probably gained a lot of insight into maintenance. Speaking broadly, what did you find out?

Dave: Of course I found out about software maintenance factors, characteristics, relationships, and so forth, as documented in the various reports and publications related to the RAMSS. However, there are many foundation truths buried in all the data that are perhaps even more important. I can't begin to itemize or discuss many of these, but I'll touch on a few thoughts, if you like.

Ned: Yes please, pick one and tell me about it.

Dave: Here is one:

Software support/maintenance is not a well-understood activity.

Perhaps the most important result for me was that software support is not well understood—it is much more complex than most people realize. Perhaps this is the case because we define software support as all those activities that occur after the software is delivered. This definition pretty well leaves the door open for including any activities over a rather extended period of time.

The interesting part of this observation is that I came to this conclusion in about 1977. In spite of all my work, and the work of many others, this observation appears to be just as valid today! Occasionally I see some glimmer of hope when there seems to be a flurry of concern over the software support cost or possible business impact due to legacy systems that need to be changed. Consider the year 2000 millennium problem. But, in almost the same breath, I hear that the way to reduce software support cost is to 'get it right the first time' or that 'the way to solve legacy system support is to get rid of all legacy systems within the near future'.

If we get it right the first time, then perhaps there is no need to change

anything, so software support cost for changes can go to zero? Is this vision realistic? Software was created so there could be some flexibility provided for change. That is why systems are becoming more and more dependent on software functionality. No, we must realize how to build software for supportability, and what continued supportability means to development rather than the reverse. We must understand the software support activity within the context of system support and require software to be delivered in a supportable state. This will require funding for both new and existing systems that is targeted to improving software support characteristics. The payback will be in software reuse.

Is it reasonable to get rid of legacy systems? There are certainly some legacy systems that need to be replaced—but isn't that transition process part of the software support activity? I do not know of anyone who has successfully developed a process, with quantitative measures, to determine when software is beginning to outlive its usefulness. Is anyone studying the retirement and transition of software functionality in a systematic manner? Are funds being allocated to conduct this analysis in a timely manner so extreme actions (such as meeting the upcoming year 2000 millennium crisis is causing) are not needed? There are isolated successes, but little progress.

Ned: That is a very important observation, and is consistent with my observations too. Assuming we are both right on it, do you see any management implications?

Dave: We derived a profile of effort by change types over time based on the maintenance data—a profile that is slightly different from the change type percentages suggested by Lientz and Swanson (1980) in their landmark publication, and earlier previewed with Tompkins (Lientz, Swanson andompkins, 1978). I show the comparison in Table 2.

Ned: What do you make of the numbers in your Table 2?

Dave: The systems we studied in the RAMSS project were at or within a few years of their initial operational capability (deployment). Even though the type of data we collected focused on the *number* of change requests, since individual change request effort was somewhat limited, we did collect some other interesting information. First, for the changes that occurred for our systems, whether the change was corrective, perfective or adaptive, the change type was not correlated

Table 2. Maintenance effort by type

Maintenance type	Percentage of effort reported by:			
	RAMSS	Lientz and Swanson[1]	Lientz, Swanson and Tompkins[2]	Dekleva[3]
Corrective	70%	21·7%	17·4%	27·7%
Perfective	25%	51·3%	60·3%	62·2%
Adaptive	5%	27·0%	18·2%	9·9%
Total	100%	100%	100%	100%

Sources: [1]Lientz and Swanson (1980, pp. 68, 73); [2]Lientz, Swanson and Tompkins (1978); [3]Dekleva (1992).

to the effort to accomplish the change. This means generally, that we could not say that it took more effort to implement an enhancement change than a corrective. The effort to accomplish the change was correlated more with the complexity of the change request and with the priority of the change request (in other words—shortness of schedule). In my experience, more corrective maintenance occurs towards the initial release. As systems mature, the corrective maintenance percentage declines, and may bottom out around 20% as the perfective and adaptive maintenance activities dominate. Then as systems age, perfective and adaptive maintenance eventually get nearly crowded out by corrective maintenance. I believe Lehman and Belady (1985) have reported a somewhat similar general pattern.

Through further analysis, I concluded that there really was no significant anomaly in Table 2 apparent differences in percentages. Instead, the differences offer insight into the funding for software maintenance activity. In the time period near the initial delivery of the software, there will be a significantly larger percentage of corrective maintenance actions, since the software defects in operational use will be discovered at that time and the priority for change will be on making sure the software works as required. As these problems get resolved, there will be more changes for enhancements since additional functionality and performance will be required. As equipment is replaced by better versions and interfaces change, adaptations and even more enhancements will be required. At some point in time, the enhancements will be the primary focus of maintenance activity, just as indicated by Lientz and Swanson (1980, pp 176–177). As the software evolves over time, with many enhancements and adaptations, the number of corrective maintenance actions (due to imperfect implementation of adaptations and enhancements) will begin to increase. At some point in time, these corrective maintenance actions will again begin to drive the maintenance activity, and eventually will dominate to such a point that the software begins to 'die'. At this point, or sometime before, the software support retirement planning should begin. But we saw none of this in the systems we examined.

Ned: How about a second of your foundation truths?

Dave: Here is a second one:

> *Quantitative methods are not being used to make software support/maintenance decisions.*

Decisions on software support and the specific software maintenance activities were not and still are not, being made on any quantitative basis. Support centres knew that they had too much to do for the resources available, but still failed to define processes that could provide the needed justification measures. Software support was treated as a people-intensive activity that depended almost solely on the capabilities of individuals. And, frequently those individuals moved on to other opportunities. Even when methods such as the RAMSS are developed, at a reasonably significant cost, and with a reasonable base of data, the methods are

not used. Even though AFOTEC commissioned the creation of the RAMSS, AFOTEC has used only parts of the methodology.

I have used the complete methodology several times in subsequent tasks for AFOTEC and other organizations, but as far as I know, only the earlier product, process and environment evaluation methods were used extensively. The product evaluation method was actually adopted by other military services. In AFOTEC's defence, the quantitative measures were not validated and there was organizationally much to be desired in having some wriggle room with fuzzy data. However, the complete framework of factors and characteristics in the RAMSS is founded on statistically significant findings drawn from the actual maintenance and evaluation data collected.

Ned: That foundation truth feels like human nature showing its colours. How about a third foundation truth relevant to software maintenance?

Dave: Here is a third one:

> *Software support/maintenance is not just an extended form of software development.*

This is one of my favourite soap boxes. Software support/maintenance is not the same as an extended software development. Software support is a unique activity of its own, deserving of study and research related to characteristics that are not the same as in software development. Of course, software development and support are forever linked together as a single life cycle process, where development activities have significant and lasting impact on support. However, there are many aspects to software support that require processes very different from development, including the rather significant concept of customer service and responsiveness to change requests. In development, we capture this concept as perturbations to requirements, and try our best to eliminate or limit such unwanted, negative consequences of 'customer interference'. In support, changes for enhancements and adaptations (effectively changes to requirements) are expected from the customers—and even are the basis for major business decisions and financial success. Furthermore, since software support occurs over quite a variety of environmental conditions, and in particular, an extended period of time, the future shock for software support is much more significant than for development and must be a part of the software support management function.

Ned: In my observation, your third truth also applies to the management of software maintenance. Can you offer a fourth foundation truth?

Dave: Yes, here is a fourth one:

> *Software support/maintenance is still recognized as an important activity and all is not lost.*

I have also seen this one from my related work in software quality and software

reliability. I have used this foundation to investigate the area of software logistics, integrating software support with logistic support methods. I am currently chair of the Society of Automotive Engineers SAE G-11 Reliability, Maintainability, Supportability, Logistics Software Committee, or SAE G-11 Software Committee for 'short'. We are developing a Software Supportability Task Guide that should be useful to software managers, developers and supporters. AIR5121 (1997) gives an introduction. I have been very encouraged by some initial responses, from European and United States organizations, about the need and timeliness of such information. Of course, the activity is mostly volunteer (with many international participants), and very little direct funding. Hopefully, everyone will realize there is no silver bullet, even for software support, and that it is a lot of hard work.

Ned: Thanks for sharing those insights. We have covered a lot of technical material, but we have not covered one topic, Dave Peercy himself. The resume (biography) at the start of this interview tells formal things, like employment and publications. What it does not give is a picture of the person, Dave Peercy. Could you please tell me about Dave?

Dave: Till this question, Ned, I felt fairly comfortable! It is difficult to talk about myself because I have so many different interests—and everyone may not be interested. I love my sports and music. I've played numerous sports over the years, a couple at collegiate level, and have continued to coach my two sons over the years, mostly in soccer (which is one of the few sports I never played—isn't that typical?). I've played piano and clarinet for my whole life and from my early classical upbringing (my father and mother were both in music) have come also to appreciate jazz, country and even soft rock. Still not too enthusiastic about the noisy sounds, but am getting more tolerant as I age and can't hear as well. My wife has put up with way more than necessary to accommodate all my professional and hobby interests and has been a wonderful part of my life. Throughout my life I've been involved in church work, primarily through my music, and this has been a significant part of our family activities. For my professional activities I enjoy problem solving, which is why I must have gravitated to mathematics for the structural foundation and software for the problems!

Ned: Dave, I am glad to get to know you better, and thank you for taking the time to share your insights and experiences about software maintenance.

Dave: My pleasure.

References

AFSC (1988) 'Software risk abatement', AFSC/AFLC Pamphlet 800–45, U.S. Air Force Systems Command, Rome, NY. pp 24.

AIR5121 (1997) 'Software supportability—an overview', Aerospace Information Report 5121, Society of Automotive Engineers, Detroit, MI, 14 pp.

Boehm, B., Brown, J. R., Kaspar, H., Lipow, M., MacLeod, G. J. and Merritt, M. (1978) *Characteristics of Software Quality*, Elsevier North-Holland Publishing Co., New York, NY. 216 pp.

Boehm, B. (Ed) (1989) *Software Risk Management*, IEEE Computer Society Press, Los Alamitos, CA. 450 pp.

Dekleva, S. M. (1992) 'Software maintenance: 1990 status', *Journal of Software Maintenance: Research and Practice*, **4**(4), 233–247.

DeMarco, T. (1982) *Controlling Software Projects: Management, Measurement & Estimation*, Prentice-Hall, Englewood Cliffs, NJ. 284 pp.
Fenton, N. (1991) *Software Metrics: A Rigorous Approach*, Chapman & Hall, London. 349 pp.
Finkelstein, L. (1982) 'What is not measurable, make measurable', *Measurement and Control*, **15**(3), 25–32.
Haimes, Y. Y., Shima, T., Tarvainen, K. and Thadathil, J. (1990) *Hierarchical-multiobjective Analysis of Large Scale Systems*, Hemisphere Publishing Corp., Bristol, PA., 333 pp.
Humphrey, W. (1990) *Managing the Software Process*, Addison-Wesley Publishing Co., Reading, MA. 512 pp.
Kaplan, J. (Ed) (1992) *Barlett's Familiar Quotations*, Little, Brown, & Co., Boston, MA. 1405 pp.
Lehman, M. M. and Belady, L. A. (1985) *Program Evolution—Processes of Software Change*, Academic Press, New York, NY. 560 pp.
Leung, H. K. N. (1996) 'A risk index for software producers', *Journal of Software Maintenance: Research and Practice*, **8**(5), 281–308.
Lientz, B. P. and Swanson, E. B. (1980) *Software Maintenance Management*, Addison-Wesley Publishing Co., Reading, MA. 214 pp.
Lientz, B. P. Swanson, E. B. and Tompkins, G. E. (1978) 'Characteristics of application software maintenance', *Communications of the ACM*, **21**(6), 466–471.
McCabe, T. (1976) 'A complexity measure', *Transactions on Software Engineering*, **SE–2**(4), 308–320.
McCall, J., Richards, P. and Walters, G. (1977) 'Factors in software quality', Volumes I, II and III of Technical Report RADC-TR-77-369, Rome Air Center, Rome, NY.
Musa, J. D., Iannino, A. and Okumoto, K. (1987) *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill Book Co., New York, NY, 621 pp.
Myerson, M. (1996) *Risk Management Processes for Software Engineering Models*, Artech House, Inc., Norwood, MA, 272 pp.
Parnas, D. L. (1979), 'Designing software for ease of extension and contraction', *Transactions on Software Engineering*, **SE–5**(3), 128–137.
Peercy, D. E. (1981) 'A software maintainability evaluation methodology', *Transactions on Software Engineering*, **SE–7**(4), 343–351.
Peercy, D. E. (1994) 'Applying the logistics discipline to software support concerns', *Logistics Spectrum*, **28**(4), 44–48.
Peercy, D. E. (1996) 'Software supportability—a logistics concern', *Logistics Spectrum*, **30**(6), 8–10, 17.
Peercy, D. E., Huebner, W., Estill, M. and Wu, J. (1985) 'Software supportability risk assessment in OT&E: historical baselines for risk profiles', Volumes I and II, BDM Technical Report BDM/A-85-0510, BDM Corp., Albuquerque, NM. 600 pp.
Peercy, D. E. and Swinson, G. (1983) 'A software support facility evaluation methodology', in *Proceedings of Symposium on Application and Assessment of Automated Tools for Software Development*, IEEE Computer Society Press, Los Alamitos, CA, 5 pp.
Peercy, D. E., Tomlin, E. and Horlbeck, G. (1987) 'Assessing software supportability risk: a minitutorial', in *Proceedings of the Conference on Software Maintenance—1987*, IEEE Computer Society Press, Los Alamitos, CA, pp. 72–80.
Porter, W. T. and Perry, W. E. (1987) *EDP Controls and Auditing*, Fifth Edition, PWS-KENT Publishing Co., Boston, MA, 617 pp.
Rowe, W. D. (1988) *Anatomy of Risk*, revised edition, Krieger Publishing Co., Melbourne, FL. (Original edition published 1977 by John Wiley & Sons, Inc., New York, NY, with the title *An Anatomy of Risk.*)
Sherer, S. A. (1992) *Software Failure Risk: Measurement and Management*, Plenum Press, New York, NY. 276 pp.
Sisti, F. and Sujoe, J. (1994) 'Software risk evaluation method version 1.0', Technical Report SEI-94-TR-19, Software Engineering Institute, Pittsburgh, PA. 94 pp.
Thayer, P. (1978) 'Software Maintainability Evaluation Methodology', AFOTEC Report, Air Force Operational Test and Evaluation Center, Kirkland Air Force Base, NM. 31 pp.

